ELSEVIER
Theoretical Computer Science 172 (1997) 233–254
Theoretical Computer Science

Fugitive-search games on graphs and related parameters[1]

Nick D. Dendris, Lefteris M. Kirousis*, Dimitrios M. Thilikos

Department of Computer Engineering and Informatics, University of Patras, Rio, 265 00 Patras, Greece
Computer Technology Institute, P.O. Box 1122, 261 10 Patras, Greece

Received August 1994; revised May 1996
Communicated by G. Ausiello

Abstract

The goal of a fugitive-search game on a graph is to trap a fugitive that hides on the vertices of the graph by systematically placing searchers on the vertices. The fugitive is assumed to have complete knowledge of the graph and of the searchers' moves, but is restricted to move only along paths whose vertices are not guarded by searchers. The search number of the graph is the least number of searchers necessary to trap the fugitive. Variants of the fugitive-search game have been related to important graph parameters like treewidth and pathwidth. In this paper, we introduce a class of fugitive-search games where the searchers do not see the fugitive and the fugitive can only move just before a searcher is placed on the vertex it occupies. Letting the fugitive's speed (i.e. the maximum number of edges the fugitive can traverse at a single move) vary, we get different games. We show that if the speed of the fugitive is unbounded then the search number minus 1 is equal to the treewidth of the graph, while if the speed is 1 then the search number minus 1 is equal to a polynomially computable graph parameter which is called width, or alternatively linkage, and is studied in the context of the Constraint Satisfaction and Boolean Satisfiability Problems. We also show that in the above two cases, the search number remains the same even if we consider only search strategies that at every step further restrict the fugitive's possible resorts (this monotonicity property is usually expressed as: "recontamination does not help"). Moreover, we give an equivalent characterization of the search number for any given fugitive speed in terms of an elimination ordering of the vertices of the graph. Using this characterization, we show that for any graph, if the length of any chordless cycle is bounded by a constant s ($s \geqslant 3$), then the treewidth of the graph plus 1 is equal to the search number for fugitive speed $s-2$.

1. Introduction

In the sequel, let $G=(V,E)$ be a connected, undirected graph without multiple edges or self-loops. The fugitive-search game was introduced by Parsons [30, 31] (see also

* Corresponding author. E-mail: kirousis@fryni.ceid.upatras.gr.
[1] This research was partially supported by the European Union ESPRIT Basic Research Projects ALCOM II (contract no. 7141), GEPPCOM (contract no. 9072) and Insight II (contract no. 6019).

PII S0304-3975(96)00177-6

[10]). In the original version of the game, the graph is thought of as a system of tunnels where an omniscient fugitive with unbounded speed is hidden. The object of the game is to trap the fugitive using searchers. A searcher can either be placed on an arbitrary vertex of the graph, or be removed from the graph, or slide along an edge. The fugitive cannot go through a vertex guarded by a searcher; it is trapped once a searcher is placed on the vertex it currently occupies and there is no place for it to go. Also, the searchers cannot see the fugitive. The fugitive being omniscient means that it a priori has complete knowledge of the graph and of the searchers' moves. It exploits this knowledge to move along unguarded paths to locations where it is harder to get trapped. The goal of the game is to trap the fugitive using the least possible number of searchers. Megiddo et al. [28] showed that computing the search number is an NP-hard problem. The fact that it actually belongs to the class NP follows from an important monotonicity result of LaPaugh [24] (see also [5]) stating that excluding search strategies which give the fugitive the possibility of "recontaminating" the graph, namely visiting an already searched vertex, does not increase the search number (in short: "recontamination does not help to search the graph"). Therefore, only monotone search strategies, i.e. strategies where the searched portion of the graph is never decreased, need to be considered.

A variant of the search game, called node-search game, was introduced in [21]. In this variant, searchers can only be placed on or removed from the vertices of the graph (no sliding is allowed). The fugitive resides on a vertex and is allowed to move from one vertex to another along unguarded paths. Again, the fugitive is assumed to have unbounded speed and be omniscient: the searchers can be placed on any vertex, but they cannot see the fugitive. It turned out that for this variant, which also has the monotonicity property of the original version [21], the search number is equal to the interval thickness of the graph (i.e. the size of the smallest max-clique in any interval supergraph of G; see [20]) and therefore to the pathwidth of the graph plus 1 (see [30]). Results relating search number to other graph parameters can be found in [14, 19, 29].

Seymour and Thomas [33] introduced still another variant of the fugitive-search game. Their setting differs from node-search in that the fugitive is *visible*. That is, at every stage of the search the searchers can see the vertex of the graph where the fugitive resides and use this knowledge to reassign their positions accordingly. This additional ability of the searchers introduces a kind of interaction between the searchers and the fugitive. Seymour and Thomas showed that the search number for this variant is equal to the treewidth plus 1 (for a survey of results related to treewidth see [7, 23]). They also showed that the monotonicity property still holds, i.e. excluding search strategies that allow the fugitive to visit a searched vertex is of no help to the fugitive. They proved the monotonicity property by showing that if for a given number of searchers the fugitive has an escape strategy, then there is a nice escape strategy, i.e. there is a collection of sets of vertices that offer a resort to the fugitive in the sense that there always exists the possibility for the fugitive to move from any such set of vertices to another one independently of the location of the searchers. The existence of such a

resort is proved using ideas on obstruction sets (see [32]). Bienstock [3] gives a survey of the related results.

In this paper, we examine search games where the searchers are always assumed *not* to be able to see the fugitive. Again, the fugitive resides on vertices, moves along unguarded paths and is supposed to be omniscient. As for the searchers, similarly to [20] and not to [30, 31], they can only be placed on, or removed from, the vertices of the graph, one at a time, with the goal to eventually trap the fugitive. However, the mobility of the fugitive is restricted: we assume that the fugitive is *inert*, i.e. it only moves just before a searcher visits the vertex it occupies (given of course that there is a vertex that can be reached via an unguarded path; otherwise the fugitive is trapped). Formal definitions are given in the next section. We prove that this inert-fugitive search game has the monotonicity property (i.e. recontamination does not help) and that the corresponding search number is equal to the treewidth plus 1. From this and the results of [33], it follows that the search numbers for an inert fugitive on one hand and for a visible fugitive, on the other, coincide. This relates the effect of a restriction on the mobility of the fugitive (inertness) with the effect of an enhancement of the abilities of the searchers (ability to see the fugitive). In contrast to this, based on results of [20, 29] one can easily show (see Section 2, Theorem 2) that without the inertness restriction on the fugitive the corresponding search number, being equal to the node-search number [21], is equal to the pathwidth plus 1. It must be pointed out that our proof of the monotonicity property for the inert-fugitive game makes use of the result by [33] about the existence of a "resort" for an escaping fugitive. However, our proof relating the search number with the treewidth is completely different from the corresponding one in [34] (as we already pointed out, the latter refers to an interactive game, where, rather than restricting the mobility of the fugitive, the searchers are assumed to have the additional ability of "vision").

We also examine search games where the fugitive, besides being inert, is further restricted to have speed equal to a given number s, i.e. the number of unguarded edges it can traverse at each move is at most s. If $s = n - 1$ (n is the number of vertices of the graph) we say that the speed is unbounded, since, in this case, the fugitive can tranverse any unguarded path. We thus obtain a class of fugitive-search games parametrized in terms of the speed of the fugitive.

We show that if the speed is 1 then the monotonicity property holds (i.e. recontamination does not help) and moreover, the search number minus 1 is equal to the width (also known as linkage), a graph parameter studied in the context of the Constraint Satisfaction and Boolean Satisfiability Problems (see e.g. [11–13, 16, 17, 25, 26]). Despite the etymological affinity, width is polynomially computable for arbitrary graphs, whereas treewidth and pathwidth are NP-complete. The sequential and parallel complexity of width have been extensively studied in [1, 22]. To define the width of a graph consider a layout of the graph and let the backdegree of a vertex v be the number of vertices preceeding v in the layout that are adjacent to v; the minimum over all layouts of the maximum backdegree of any vertex of the graph is the width of G. It is known that the width of G is equal to the maximum min-degree of any subgraph

of G. Certain classes of graphs with bounded width are advantageous for applying backtracking, the classical method to solve the Constraint Satisfaction Problem (see [25]). In analogy to the classical definitions of pathwidth and treewidth, we also give a new characterization of the width in terms of a decomposition of the graph into sets of vertices.

The above characterizations of the search numbers for fugitives with differing abilities, but with identical rules for searchers, offer a uniform game-theoretic approach to pathwidth, treewidth and width.

Other variants of the fugitive-search game with mobility restrictions were used by Franklin et al. [15] to model issues of privacy in distributed systems. However, they only consider variations on the mobility settings of the searchers, rather than the fugitive (for them the searchers represent bugs eavesdropping in a network, whereas the fugitive represents the information to be captured).

In the last section, we give a result implying interesting algorithmic properties. We characterize the search number for an inert fugitive with a given speed in terms of an elimination ordering. Using this characterization, we prove that for any graph whose largest chordless cycle is at most $s+2$, the treewidth plus 1 is equal to the search number for an inert fugitive with speed s. This is a new characterization for the treewidth of a graph in terms of the length of its longest chordless cycle. Intuitively, this fact asserts that for graphs with small chordless cycles, even if we decrease the speed of the fugitive, this does not offer any advantage of the searchers. Our characterization was used in [9], to investigate the complexity of treewidth and the existence of small separators in special classes of graphs. We finally mention that, if the speed is 1, then this result implies that for triangulated graphs, the treewidth is equal to the width (however, this equality is an easy corollary of extant results).

2. Formal definitions and results on pathwidth

A *search* on a graph $G=(V,E)$ is a sequence $S_0,\ldots,S_r$ of sets of vertices ($S_i \subseteq V$, $i=0,\ldots,r$) such that $S_0=\emptyset$ and for all $i=1,\ldots,r$ the symmetric difference of the sets S_i and S_{i-1} has cardinality 1. Intuitively, set S_i contains the vertices holding a searcher at step i of the search, and at each step of the search, either one searcher is added on or one searcher is removed from the graph.

The *search number of a search* is the maximum of the cardinalities of the sets S_i, $i=0,\ldots,r$.

Let $\mathcal{S}=\{S_i,\ i=0,\ldots,r\}$ be a search. For $i=0,\ldots,r$, we inductively define the set of *free locations* F_i for an *agile fugitive with unbounded speed* as follows:

- $F_0=V$.
- For $i=1,\ldots,r$, let $F_i=(F_{i-1}-S_i)\cup\{v\in V-S_i$: there is a path from a vertex $u\in F_{i-1}$ to v whose vertices except u belong to $V-S_i\}$.

Intuitively, after the ith step of the search, the fugitive can be at *any* of the vertices of F_i. Being omniscient, after each step of the search, it chooses a most advantageous

location in F_i. Intuitively also, $V - S_i$ is the set of unguarded vertices. The fugitive moves along unguarded paths to vertices that have possibly admitted a searcher in the past. The fugitive is agile in the sense that it has the ability to move whenever there appears an unguarded path that starts from its current location (also see below the definition of the inert-fugitive game). It is easy to see that the search game for an agile fugitive as defined here is exactly the same as the node-search game introduced in [21].

The set of free locations for an *inert fugitive with unbounded speed* is defined as follows:

- $F_0 = V$.
- For $i = 1,\ldots,r$, let $F_i = (F_{i-1} - S_i) \cup \{v \in V - S_i$: there is a path from a vertex $u \in F_{i-1} \cap (S_i - S_{i-1})$ to v whose vertices except u belong to $V - S_i\}$.

Intuitively, an inert fugitive is allowed to move only when a searcher is about to be placed on the vertex it occupies. This is so because the fugitive can move away from a vertex u only if $u \in F_{i-1} \cap (S_i - S_{i-1})$. Notice also that, for an inert fugitive, if $S_i \subseteq S_{i-1}$ or even if $S_i \subseteq V - F_{i-1}$, then $F_i = F_{i-1}$, i.e. when searching for an inert fugitive, if we remove a searcher from the graph or place a searcher to a vertex not in the set of the fugitive's free locations, this does not cause the fugitive to move.

Finally, if n is the number of vertices of the graph and $1 \leqslant s \leqslant n-1$ is an integer, the set of free locations for an *inert fugitive with speed s* is defined as follows:

- $F_0 = V$.
- For $i = 1,\ldots,r$, let $F_i = (F_{i-1} - S_i) \cup \{v \in V - S_i$: there is a path of length at most s from a vertex $u \in F_{i-1} \cap (S_i - S_{i-1})$ to v whose vertices except u belong to $V - S_i\}$.

Intuitively, an inert fugitive with speed s behaves exactly as an inert fugitive with unbounded speed except that it cannot traverse a path of length more than s.

Given the type of the fugitive, a search $\mathcal{S}$ is *complete* if $F_r = \emptyset$. For each type of fugitive, the corresponding *search number of the graph* is the minimum search number over all searches which are complete with respect to this type of fugitive.

For all types of fugitives, a search is called *monotone* if $\forall i = 1,\ldots,r$, $F_i \subseteq F_{i-1}$. Notice that for a monotone search and for all types of fugitives, $F_i = F_{i-1} - S_i$. Intuitively, a search is monotone if it does not allow recontamination.

For each type of fugitive, the corresponding *monotone search number of the graph* is the minimum search number over all monotone searches which are complete with respect to this type of fugitive.

Fig. 1 depicts a graph whose search number for an agile fugitive with unbounded speed is 6; its search number for an inert fugitive with unbounded speed is 4; and its search number for an inert fugitive with speed 1 is 3. We now informally describe how these numbers are attained.

(1) For a fugitive with unbounded speed (agile or inert), start searching the graph from left to right, keeping as "attack front" searchers on u_1, u_2 and u_3; then on u_4, u_5 and u_6; and then on u_7, u_8 and u_9 (four searchers are required, the extra one for the transition from one "front" to the next). Then proceed to search each of the two branches of the graph.

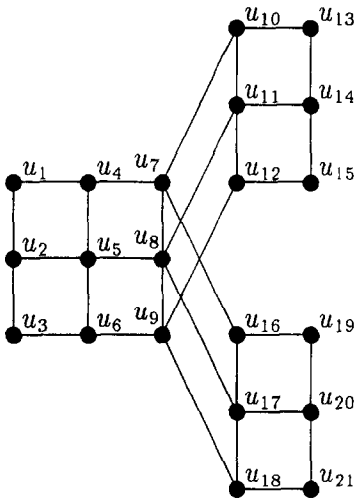

Fig. 1. An example graph which for an agile fugitive with unbounded speed has search number 6; for an inert fugitive with unbounded speed has search number 4; and for an inert fugitive with speed 1 has search number 3.

(a) If the fugitive is agile, during the search of the upper branch, keep watches on the three vertices u_7, u_8 and u_9 to guard off the fugitive from the lower branch. As it can be easily seen, only three extra searchers are enough to search the upper branch (the lower branch is easier), thus the total number of searchers needed is 6.

(b) If the fugitive is inert, there is no need for a watch; four searchers altogether suffice.

(2) In the case of an inert fugitive with speed 1, place searchers on u_1, u_2 and u_3, then remove the searcher residing on u_3 and place it on u_5 (since the speed is 1, the fugitive cannot go to u_3): then, in the same way, move the searcher residing on u_2 to u_4 and then back to u_3: then move the searcher from u_1 to u_6: finally move the searcher from u_3 to u_4. Thus the transition from one "attack front" to the next one has been attained with only 3 searchers. Continue like this, to search all possible vertical triples of vertices.

Theorem 1. *For a graph G and for an agile fugitive with unbounded speed, the monotone search number of G is equal to the search number of G.*

Proof. The theorem is a restatement of the corresponding result in [21]. The proof in [21] is obtained by reduction to *edge-searching*, the original variant of the fugitive-search game; the monotonicity of this variant was first proved by LaPaugh [24] (see also [4, 5]). □

Notice that in the case of an inert fugitive, a *monotone* search may entail re-insertion of a searcher on a vertex that has already been visited by a searcher (of course, by monotonicity, the fugitive cannot be on such a vertex). The monotonicity property

for such searches guarantees only that the fugitive cannot visit an already searched vertex (and not that a searcher is never re-inserted on a vertex that has previously admitted a searcher). For example, to search the graph of Fig. 1 for a fugitive with unbounded speed using only 4 searchers, it is necessary, at some steps of the search, to re-insert searchers to already searched vertices. However, for an agile fugitive, it is not only known that recontamination does not help, but also that searcher re-insertion is unnecessary [20].

We now give, for completeness, the definition of pathwidth.

Definition 1. A *path-decomposition* of $G = (V,E)$ is a class $\{X_i: i = 1,\ldots,r\}$ of subsets of V that satisfies the following conditions:

1. $\bigcup_{i=1}^{r} X_i = V$.
2. $\forall\{u,w\} \in E$, $\exists i$: $u, w \in X_i$.
3. $\forall i,j,k$, if $1 \leqslant i \leqslant j \leqslant k \leqslant r$, then $X_i \cap X_k \subseteq X_j$.

The *pathwidth* of a path-decomposition $\{X_i: i = 1,\ldots,r\}$ is defined to be $\max_{1 \leqslant i \leqslant r} |X_i| - 1$. The pathwidth of G is the minimum pathwidth of any path-decomposition of G.

Finally, we mention:

Theorem 2. *The pathwidth of G incremented by* 1 *is equal to the search number of G for an agile fugitive with unbounded speed.*

Proof. Kirousis and Papadimitriou [20] prove that the node-search number of G is equal to the interval thickness of G (interval thickness is defined to be the smallest max-clique in any interval supergraph of G). Möhring [29] shows that the pathwidth plus 1 is also equal to the interval thickness. The theorem immediately follows from these results and from the fact that the node-search game is the same as the agile-fugitive search game. □

Fig. 2(a) depicts in detail a monotone search for an agile fugitive with unbounded speed in a three-dimensional cube. Fig. 2(b) depicts an optimal path-decomposition for the graph. Notice that the pathwidth of this graph is 4.

3. Inert-fugitive game and treewidth

In this section, we show that the search number for an inert fugitive with unbounded speed is equal to the treewidth of the graph incremented by 1. We also show that for this type of fugitive, the monotone search number is equal to the (nonmonotone) search number. The proof of this monotonicity result depends on the existence of a screen, a notion introduced by Seymour and Thomas [33]. Screens are obstructions for graphs with small treewidth.

For completeness, we first give the definition of treewidth.

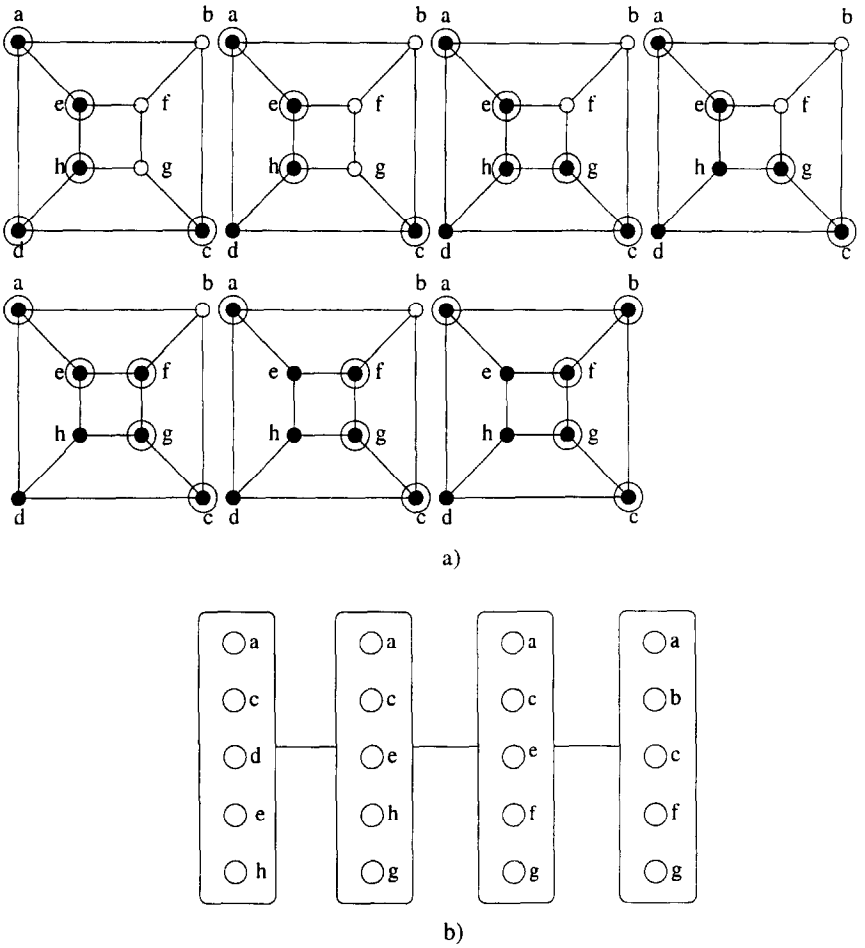

Fig. 2. (a) The steps of an example monotone search of the three-dimensional cube for an agile fugitive with unbounded speed. The search steps are drawn left to right, top to bottom. A circle around a vertex indicates that a searcher resides on it; cleared vertices are drawn filled. (b) An optimal path-decomposition for the three-dimensional cube.

Definition 2. A *tree-decomposition* of $G = (V,E)$ is a pair $(\{X_i: i \in I\}, T)$ where $\{X_i: i \in I\}$ is a collection of subsets of V and $T = (I,F)$ is a tree having the index set I as set of vertices so that the following conditions are satisfied:

1. $\bigcup_{i\in I} X_i = V$.
2. $\forall\{u,w\} \in E$, $\exists i \in I$: $u,w \in X_i$.
3. $\forall i,j,k \in I$: if j is on a path in T from i to k then $X_i \cap X_k \subseteq X_j$.

The *treewidth* of a tree-decomposition $(\{X_i: i \in I\}, T)$ is defined to be $\max_{i\in I} |X_i| - 1$. The *treewidth* of G is the minimum treewidth of any tree-decomposition of G.

Treewidth has many equivalent characterizations (see e.g. [2, 6]). We are going to use one expressed in terms of elimination orderings of graphs. An *elimination ordering* of

a graph $G = (V,E)$ is an ordering $\pi = (v_1, \ldots, v_n)$ of the vertices of G ($n = |V|$). The graphs generated during an elimination of the vertices of G according to π are defined to be: $G_1 = G$ and G_{i+1} = the graph obtained from G_i by deleting vertex v_i and adding new edges (if necessary) so that all pairs of neighbors of v_i in G_i are adjacent in G_{i+1}. Obviously, G_{n+1} = empty graph. The *dimension of* v_i with respect to π is the degree of v_i in G_i. The *dimension of* π is the maximum dimension of any $v_i \in \pi$, and finally the *elimination dimension* of G is the minimum dimension of any elimination ordering of G. The following result can be found in [2]:

Theorem 3. *The treewidth of a graph is equal to its elimination dimension.*

Given an elimination ordering $\pi = (v_1, \ldots, v_n)$ of G, it is convenient to define the *support* of a vertex v_i to be the set of vertices v_j with $j > i$ that are connected to v_i by a path in G whose vertices except its endpoints v_i and v_j (if any) precede v_i in π.

We need the following easy technical lemma.

Lemma 1. *Let $\pi = (v_1, \ldots, v_n)$ be an elimination ordering of G. Then for every v_i, the support of v_i is equal to the set of neighbors of v_i in G_i and therefore the cardinality of the support of v_i is equal to the dimension of v_i with respect to π.*

Proof. Let u be a vertex after v_i in π which is connected to v_i via a path of nodes preceding v_i in π. Let v_m be the first with respect to π internal vertex of this path. By the definition of G_{m+1}, we get that in G_{m+1} the length of the path is reduced by 1. Repeating the same argument over the length of the path, and since all internal nodes have been deleted in G_i, we get that in G_i the vertex v_i is connected to u. For the converse, observe that if u is a neighbor of v_i in G_i, then in G_{i-1}, u is connected to v_i by a path which may have v_{i-1} as an internal vertex. Repeating this argument all the way back to G, we get the required. □

We now prove the following:

Theorem 4. *The treewidth of a graph G plus 1 is equal to the monotone search number for an inert fugitive with unbounded speed.*

Proof. We first show that if the treewidth of G is k, then there is a complete monotone search for an inert fugitive with unbounded speed that uses at most $k + 1$ searchers.

By Theorem 3 there is an elimination ordering $\pi = (v_1, \ldots, v_n)$ of G which has dimension k. We construct a complete monotone search with $k+1$ searchers as follows: First, place a searcher on v_n. Thus, the fugitive may retreat at any vertex other than v_n. Assume now that all vertices after v_i in π have been visited, and the fugitive resides at some vertex in the set $\{v_1, \ldots, v_i\}$. To extend the search to clear v_i, first remove all the searchers (if any) residing on vertices of the graph, arbitrarily and one at a time. Notice that the fugitive, being inert, will not move. Afterwards, place searchers on the vertices in the support (with respect to π) of v_i. Again, this will not cause the fugitive

to move. Finally, place a searcher on v_i. As a result of placing the searcher onto v_i, the fugitive can move to any vertex he can reach from v_i. Thus, in principle, he/she could move to an already searched vertex. However, the already searched vertices that can be reached are those in the support of v_i with respect to π. Since these vertices already held a searcher before the placement of the searcher on v_i, we get that the fugitive has to escape to some vertex in the set $\{v_1,\ldots,v_{i-1}\}$, and v_i is cleared. Being the search performed from v_n to v_1, we conclude that it is complete and monotone. Also, by Lemma 1, the search uses at most $k+1$ searchers. This completes the proof of the first direction.

We now prove that if there is a complete monotone search with $k+1$ searchers, then there is an elimination ordering of dimension at most k. Order the vertices in terms of the search step that places a searcher on them for the first time; then reverse this order to get $\pi=(v_1,\ldots,v_n)$ (formally, during the search, a searcher visits v_i for the first time before a searcher visits v_j for the first time iff $i>j$). We claim that the elimination ordering thus defined has dimension at most k. Indeed, by Lemma 1, if a vertex v_i has dimension with respect to π strictly more than k, then also the support of v_i has cardinality strictly more than k. But then, since there are no more than $k+1$ searchers available, when visiting v_i with a searcher for the first time there would exist a vertex in the support of v_i not guarded by a searcher. This contradicts the monotonicity of the search. □

Fig. 3(a) depicts a monotone search for an inert fugitive with unbounded speed in the three-dimensional cube, and Fig. 3(b) depicts an optimal tree-decomposition of the same graph. Notice that the inertness of the fugitive is strongly used to search the graph with less searchers than in the pursuit of an agile fugitive.

The next step is to prove the monotonicity of the search game for an inert fugitive of unbounded speed. Crucial for the proof is the notion of screen introduced in [33]. Below we give the related definitions and then state the corresponding theorem.

Let $G=(V,E)$ be a graph and let $H_1,H_2\subseteq V$. We say that H_1,H_2 *mutually touch* if $H_1\cap H_2\neq\emptyset$ or $\exists e=\{v_1,v_2\}\in E: v_1\in H_1\wedge v_2\in H_2$.

Definition 3. A *screen* S of a graph $G=(V,E)$ is a collection $H_1,\ldots,H_r$ of nonempty subsets of V that induce connected and pairwise touching subgraphs of G. The screen S has thickness $\geqslant k$ iff $\forall X\subseteq V$ with $|X|<k$, $\exists H_i: H_i\cap X=\emptyset$.

We point out that set X in the definition above does not necessarily induce a connected subgraph to G. Notice also that the sets $H_1,\ldots,H_r$ are not required to form a cover of V.

Theorem 5 (Seymour and Thomas [33]). *Let $G=(V,E)$ be a graph. If the treewidth of $G\geqslant k$ then G has a screen of thickness $\geqslant k+1$.*

We now prove:

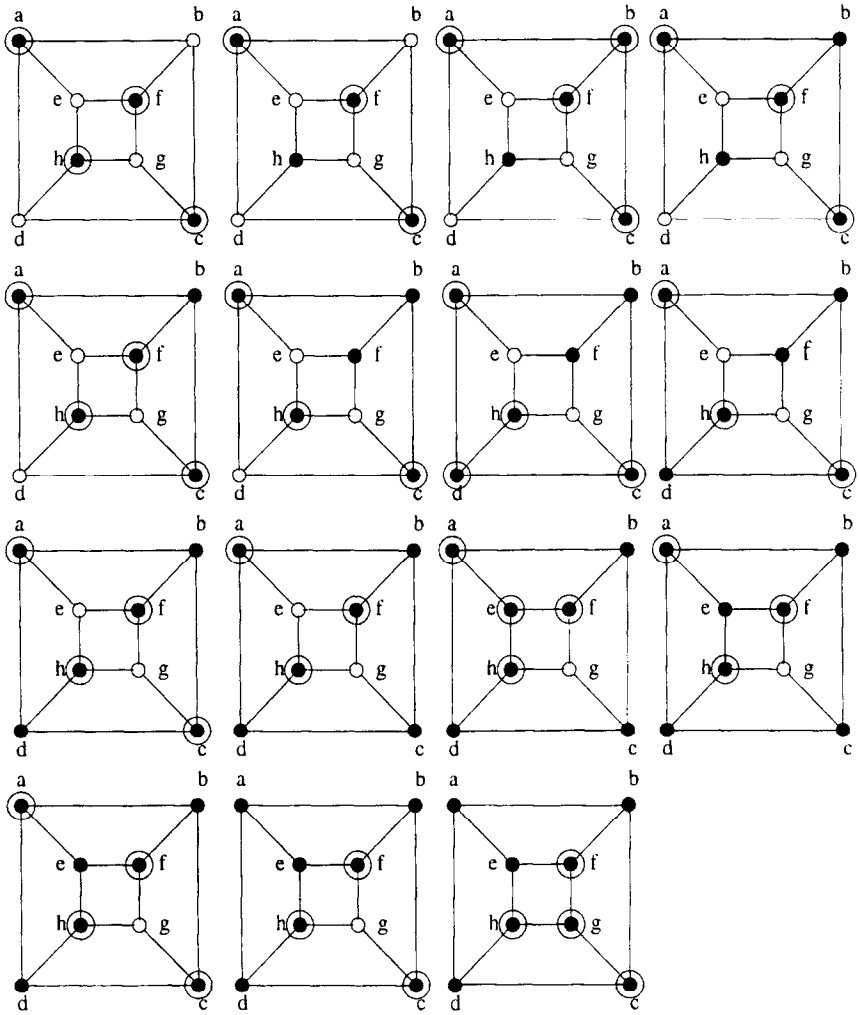

Fig. 3. (a) An example monotone search for an inert fugitive with unbounded speed for the three-dimensional cube, and (b) an optimal tree-decomposition.

Theorem 6. *If $G = (V,E)$ has a screen with thickness $\geqslant k+1$ then an inert fugitive with unbounded speed cannot be captured with $\leqslant k$ searchers (even by a nonmonotone search).*

Proof. Let $H_1,\ldots,H_r$ be a screen of G of thickness $\geqslant k+1$. Then

$$\forall X \subseteq V \text{ if } |X| < k+1, \text{ then } \exists H_i : H_i \cap X = \emptyset.$$

We will now provide a strategy for the fugitive that allows it to avoid any search that uses $< k+1$ searchers. Recall that the fugitive, being omniscient, knows in advance all the moves of the searchers. Initially, i.e. before any searchers appear on the graph, the fugitive arbitrarily selects some screen element H. Let v be the first vertex of H ever to be visited by a searcher. The fugitive chooses v as its very first location. Let X be the set of vertices occupied by a searcher immediately before v admits a searcher for the first time. Then, since the search uses less than $k+1$ searchers, $|X \cup \{v\}| < k+1$, and thus there is a screen element H' such that $H' \cap (X \cup \{v\}) = \emptyset$. Notice also that $X \cap H = \emptyset$. Thus, when a searcher is placed on v, the fugitive can escape to any vertex in H'. This is so because H and H' are connected, mutually touching and moreover, just before putting a searcher on v, H and H' carry no searcher. The fugitive, being omniscient, chooses to go to that vertex of H' that will be visited first by a searcher after the current step of the search.

Repeating this procedure, it becomes clear that the fugitive can escape being captured by $\leqslant k$ searchers forever. □

Fig. 4 depicts a screen of thickness $\geqslant 5$ for the octahedron. Notice that any placement of up to 4 searchers (not necessarily on vertices inducing a connected graph) leaves at least one searcher-free screen element.

Now, from Theorems 4–6, we get as immediate corollaries the following two results, our main results in this section:

Theorem 7. *For an inert fugitive with unbounded speed, the monotone search number of a graph G is equal to (nonmonotone) search number of G (i.e. recontamination does not help to search for such a fugitive).*

Proof. Indeed, if the monotone search number of a graph G is $k+1$, then by Theorem 4 the treewidth of G is k, so by Theorem 5 there is a screen of thickness $\geqslant k+1$ and therefore by Theorem 6 there does not exist a complete search using $< k+1$ searchers (not even a nonmonotone one). □

Theorem 8. *The treewidth of a graph G plus 1 is equal to its (nonmonotone) search number for an inert fugitive with unbounded speed.*

Proof. This follows immediately from Theorems 4 and 7. □

From the previous theorem and the results of [33], it follows that the search numbers for an inert fugitive on one hand and for a visible fugitive, on the other, coincide.

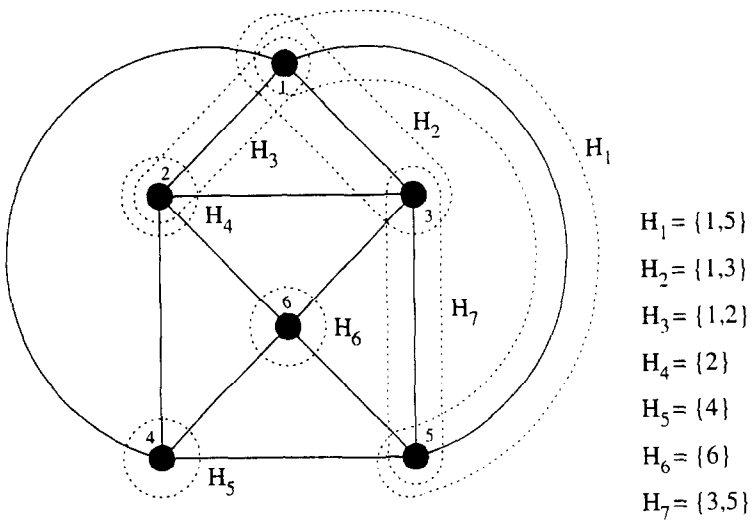

Fig. 4. A screen of thickness $\geqslant 5$ for the octahedron. The screen elements are drawn using dotted lines.

Also, as expected, the search number for an inert *and* visible fugitive (with unbounded speed), is equal to the search number for an inert only (or, alternatively, visible, only) fugitive (with unbounded speed). Indeed, by an argument similar to the proof of Theorem 6, it can easily be seen that any inert fugitive, visible or not, cannot be captured by $\leqslant k$ searchers if G has a screen of thickness $\geqslant k+1$; the claim now follows from Theorems 5 and 8. Finally, observe that the previous theorem also implies that computing the search number for an inert fugitive with unbounded speed is NP-complete (notice that not even the membership in the class NP is obvious – essentially it is a consequence of Theorem 7).

4. Unit-speed fugitive and width

In this section, we examine the search game for inert fugitives that have speed 1. We prove that the search number in this case is equal to the width, which, as we mention in the introduction, is a polynomially computable graph parameter studied in the context of the Constraint Satisfaction and Boolean Satisfiability Problems. We also prove that "recontamination does not help" to search for an inert fugitive with unit speed. The proof shows the existence of a very simple obstruction for small search number (for this type of fugitive). Finally, in analogy with the definitions of pathwidth and treewidth that we have mentioned, we give a characterization of width in terms of decomposition into sets of vertices.

We first give the related definitions.

A *layout* of a graph $G = (V,E)$ is an (ordered) n-tuple $L = (v_1,\ldots,v_n)$ of the vertices of G ($n = |V|$).

The *width* of a vertex v with respect to a layout L is the number of vertices which are adjacent to r and precede v in the layout.

The *width of a layout* L of G is the maximum width of any vertex in L.

The *width of* G is the minimum width of any layout of G.

For clarity, let us mention that we use the term *min-degree of a subgraph* H *of* G to denote the least degree of any of its vertices; the degree of a vertex is taken with respect to the subgraph.

The following theorem is proved in [16] (see also [27]).

Theorem 9. *The width of a graph is equal to the largest min-degree of any subgraph of* G.

We now prove that:

Theorem 10. *The monotone search number of* G *when searching for an inert fugitive with speed* 1 *equals its width plus one.*

Proof. Consider a layout $L = (v_1, \ldots, v_n)$, of the vertices of $G = (V, E)$ and let k be the width of L. We construct a complete monotone search for an inert fugitive with speed 1 whose search number is no more than $k + 1$. We do this as follows: Initially, place a searcher on the first vertex v_1 of L. Inductively, in order to "clear" v_{i+1}, first remove all searchers from the graph, arbitrarily and one at a time; then place a searcher on all vertices which are adjacent to v_{i+1} and precede v_{i+1} in the layout, again arbitrarily and one at a time; finally place a searcher on v_{i+1} itself. It is obvious that a fugitive that can traverse only one edge every time it moves is always forced to move further in the layout. Thus, it is finally captured. Since the width of L is k, $k+1$ searchers are used for this search. This completes the proof of the first direction.

For the converse, let $\mathcal{S}$ be a complete monotone search for an inert fugitive with speed 1, and let the search number of $\mathcal{S}$ be $k + 1$. We will define a layout L of the vertices of G whose width is at most k. Indeed, define L to be the order by which the vertices of G are visited by a searcher for the first time during $\mathcal{S}$. Since recontamination does not take place during $\mathcal{S}$, and as immediately before placing a searcher the number of searchers already on the graph is no more than k, we easily conclude that L has width at most k. □

Theorem 11. *If there is a subgraph* H *of* G *with min-degree* k, *then an inert fugitive with speed* 1 *cannot be captured using* $\leqslant k$ *searchers (even by a nonmonotone search).*

Proof. If the fugitive chooses to reside in H, any attempt to capture it with $\leqslant k$ searchers (even allowing recontamination) will be futile. Indeed, whenever the search places a searcher on the vertex of H where the fugitive is hiding (call this vertex v) there will always exist a vertex u in H which is both unguarded and adjacent to v; the fugitive can escape to u. □

From the above theorems we get:

Theorem 12. *For an inert fugitive with speed* 1, *the monotone search number of a graph G is equal to the (nonmonotone) search number of G.*

Proof. If the monotone search number is $k+1$, then, by Theorems 9 and 10, there exists a subgraph H of G with min-degree k. But then, by Theorem 11, there does not exist a complete search of G using $< k+1$ searchers (not even a nonmonotone one).

Theorem 13. *The width of a graph plus* 1 *is equal to its search number for an inert fugitive with speed* 1.

Proof. This follows immediately from Theorems 10 and 12. □

Before proceeding, let us make an observation: if we allow for an inert fugitive with speed 1 to be *visible*, then the search number does not change. Indeed, by an argument similar to the proof of Theorem 11, it can be easily seen that any inert fugitive, visible or not, cannot be captured by $\leqslant k$ searchers if G has a subgraph of min-degree k. The claim then follows from Theorems 9 and 13. As we pointed out at the end of the previous section, the same observation is true when the speed is unbounded. We do not know, however, whether this observation still holds for an arbitrary, but independent of n, value of the speed.

We now give a characterization of the width of G in terms of a decomposition of G in sets of vertices. Although this result is not immediately related to the rest of the paper, it further attests the uniformity of the notions of width, treewidth and pathwidth.

Definition 4. A *width-decomposition* of a graph $G=(V,E)$ is defined to be a class $\{X_1,\ldots,X_r\}$ of subsets of V such that the following conditions are satisfied:

1. $\bigcup_{1\leqslant i\leqslant r} X_i = V$.
2. $\forall\{u,v\}\in E,\ u\in X_{F(v)}$ or $v\in X_{F(u)}$, where $F(v)=\min_{1\leqslant i\leqslant r}\{i\colon v\in X_i\}$.

Theorem 14. *The width of a graph is equal to the minimum of the quantities* $\max_{1\leqslant i\leqslant r}|X_i|-1$ *over all its width-decompositions.*

Proof. Assume first that the width of $G=(V,E)$ is k. Then, by definition, there is a layout $L=(v_1,\ldots,v_n)$ of the vertices of G where every vertex is adjacent to at most k vertices preceding it in the layout ($n=|V|$). Define a width-decomposition X_i, $i=1,\ldots,n$ as follows: $X_i=\{v_i\}\cup\{v_j\colon j<i\wedge\{v_i,v_j\}\in E\}$. Obviously, condition 1 of the definition of a width-decomposition is satisfied. Also, it is obvious that $\max_{1\leqslant i\leqslant n}|X_i|-1\leqslant k$. Observe also that $F(v_i)=i$. To prove condition 2 of the definition of width-decomposition, let $\{v_i,v_j\}$ be an edge of the graph and assume without loss of generality that $j<i$. By construction then, $v_j\in X_i=X_{F(v_i)}$. This completes the proof of the first direction.

For the converse, assume that there is a width-decomposition $X_1,\ldots,X_r$ such that $|X_i|-1\leqslant k$, for all $i=1,\ldots,r$. We construct a layout of the vertices of the graph such that the width of every vertex in the layout is $\leqslant k$. For two vertices u and v, if

$F(u) < F(v)$, place u before v in the layout. If $F(u) = F(v)$ let the relative position of u and v in the layout be arbitrary (in other words, order the vertices by the value of F and break ties arbitrarily). We have to show that in this layout every vertex is adjacent to at most k vertices preceding it. Since $\forall i$, $|X_i| \leqslant k+1$, it is enough to show that if u percedes v and they are adjacent then $u \in X_{F(v)}$. Observe that if $F(u) = F(v)$, then obviously $u \in X_{F(v)}$. So assume that $F(u) \neq F(v)$. Therefore, since u precedes v, we have that $F(u) < F(v)$. By condition 2 of the definition of width-decomposition and since u and v are adjacent, we have that either $u \in X_{F(v)}$ or $v \in X_{F(u)}$. However, $v \in X_{F(u)}$ implies that $F(v) \leqslant F(u)$, a contradiction since we have shown that $F(u) < F(v)$. Therefore $u \in X_{F(v)}$. □

5. Elimination orderings – treewidth of graphs with chordless cycles of bounded length

In this section, we give a characterization of the monotone search number for an inert fugitive of a given arbitrary speed in terms of an elimination ordering of the graph. Using this result, we show that in the class of graphs whose largest chordless cycle has length at most $s+2$, the treewidth plus 1 is equal to the monotone search number for an inert fugitive with speed s. A corollary of this result and Theorem 10 is that for triangulated graphs, the treewidth is equal to the width, which is a polynomially computable parameter and has an NC approximation algorithm for constant approximation factors $< 1/2$ [1] (this equality for the case of unit speed also follows from extent results [34]).

As mentioned in Section 3, an elimination ordering of a graph $G = (V,E)$ is an ordering $\pi = (v_1, \ldots, v_n)$ of the vertices of the graph ($n = |V|$). Given an elimination ordering π and an integer s ($1 \leqslant s \leqslant n-1$), the graphs generated during an *s-elimination of the vertices of* G according to π are defined to be: $G_1 = (V_1, E_1)$ is the same as G; for $G_{i+1} = (V_{i+1}, E_{i+1})$, we have that $V_{i+1} = V_i - \{v_i\}$ and E_{i+1} is the set of pairs $\{u,v\}$ such that $u,v \in V_{i+1}$ and there is a path in G that connects u with v, has length at most s and all its vertices except u and v are among $v_1, \ldots, v_i$. The *s-dimension of* v_i *with respect to* π is defined to be the degree of v_i in G_i. The *s-dimension of* π is defined to be the maximum s-dimension of any of the v_is with respect to π, and finally the *s-elimination dimension of* G is the minimum s-dimension of any elimination ordering of G.

We now prove:

Theorem 15. *For any* s ($1 \leqslant s \leqslant n-1$), *the s-elimination dimension of* G *plus* 1 *is equal to its monotone search number for an inert fugitive with speed* s.

Proof. Consider a complete monotone search of G for an inert fugitive with speed s which uses $k+1$ searchers. Let $\pi = (v_1, \ldots, v_n)$ be the inverse order of the one the search places searchers on the vertices of G for the first time, i.e. v_i is before v_j in π iff the placement of the first searcher on v_j occurs before the placement of the first

searcher on v_i. We will prove that the s-dimension of π is at most k. Indeed, consider an arbitrary vertex $v_{i_0} \in \pi$ and let $v_{i_1}, \ldots, v_{i_d}$ be the set of vertices which precede v_{i_0} in π and are connected to v_{i_0} via paths of length at most s and whose internal vertices are after v_{i_0} in π. By definition, d is the s-dimension of v_{i_0}. Notice now that when a searcher was placed for the first time on v_{i_0}, there had to be a searcher in each of the $v_{i_1}, \ldots, v_{i_d}$, or otherwise recontamination would occur. Hence, the s-dimension of v_{i_0} is $\leqslant k$. This completes the proof of the first direction.

For the converse, let $\pi = (v_1, \ldots, v_n)$ be an ordering of the vertices of G, with s-dimension k. We will construct a monotone search for an inert fugitive with speed s which uses at most $k+1$ searchers. Initially, place a searcher on v_n. Assume now that we have monotonically searched all vertices after v_i, i.e. the fugitive cannot be in any v_l, $l > i$. To search v_i, first remove all searchers from the graph, arbitrarily and one at a time; then place, arbitrarily and one at a time, searchers on all v_l: $l > i$ such that there exists a path which connects v_i to v_l, has length at most s and its internal vertices are among v_m, $m < i$; finally, place a searcher on v_i. The cardinality of the set of these v_ls is no more than the s-dimension of π. Therefore, the search we defined uses at most $k+1$ searchers. Clearly also, it is a complete monotone search for an inert fugitive with speed s. □

We now state the following characterization of treewidth for the class of graphs whose chordless cycles are of bounded length.

Theorem 16. *The treewidth of a graph with no chordless cycles of length $> s+2$ is equal to its s-elimination dimension.*

To prove the above theorem, we first give some definitions and show certain lemmata.

For reasons of clarity, let us mention that we use the term *sub-tree* of a given tree to refer to a connected subgraph of the tree.

The following is proved in [18] (see also [8]).

Lemma 2. *Given a tree $T = (I,F)$, let $\mathcal{T}$ be a class of sub-trees of T such that $\forall S, S' \in \mathcal{T}$, $S \cap S' \neq \emptyset$. Then $\bigcap_{S \in \mathcal{T}} S \neq \emptyset$.*

Also, using the previous lemma the following is proved in [8].

Lemma 3. *Consider a tree-decomposition $(\{X_i,\ i \in I\}, T = (I,F))$ of a graph $G = (V,E)$. Then, for any subset K of V inducing a clique on G, $\exists i \in I$: $K \subseteq X_i$.*

Consider now an elimination ordering $\pi = (v_1, \ldots, v_n)$ of a graph $G = (V(G), E(G))$. We define below a new procedure to eliminate the vertices of G according to π. We call this procedure *tree-elimination*:

procedure tree-elimination $(H = (V(H), E(H)) : V(H) \subseteq V(G))$

(1) Let v be the first vertex of π that belongs to $V(H)$.

(2) If there exist (nonempty) connected components, say $C_1,\ldots,C_r$, of the graph obtained from H by removing v and all its adjacent (in H) vertices, then do the following steps.

(3) For each C_j, let ∂C_j be the set of vertices that are adjacent to v and, moreover, are adjacent to at least one vertex of C_j. Formally, $\partial C_j = \{u \in V(H): \{u,v\} \in E(H)$ and $\exists w \in C_j: \{u,w\} \in E(H)\}$.

(4) For each C_j, let $\bar{C}_j$ be the graph with set of vertices $V(C_j) \cup \partial C_j$ and set of edges the union of the following:

(a) the set of edges induced from $E(H)$ on $V(C_j) \cup \partial C_j$,

(b) the set of "new" edges necessary to make ∂C_j a clique.

(5) Recursively apply tree-elimination on each $\bar{C}_j, j = 1,\ldots,r$.

/*Notice that the graphs $\bar{C}_j$ are not necessarily pairwise disjoint.*/

Let $(H_l)_{l\in L}$ be the family of all graphs to which tree-elimination is recursively, applied when we run tree-elimination on G according to π. Include G in this family (say $G = H_{l_0}$). Let v_{H_l} be the first vertex in π that belongs to $V(H_l)$. Notice that for two different H_ls, the corresponding v_{H_l}s may be equal.

We give the family $(H_l)_{l\in L}$ a *rooted* tree structure as follows: The root is $G = H_{l_0}$; H_l is the parent of H_m iff when the procedure tree-elimination is recursively applied on H_l, then H_m is one of the graphs $\bar{C}_j$ defined at step 4 of this recursive call (notice that by this definition, the procedure tree-elimination will be subsequently recursively applied on H_m). We call this tree the *recursion tree.*

Define the *tree-dimension of* π to be the maximum degree of any v_{H_l} with respect to H_l. Define the *tree-elimination dimension of* G to be the minimum tree-dimension of any elimination ordering.

Lemma 4. *For any H_l, the treewidth of H_l is at most equal to the maximum of the degrees of v_{H_m}s (the degrees are taken with respect to the corresponding H_ms), over all H_ms that are either descendants of or coincide with H_l in the recursion tree. Therefore, the treewidth of G is at most equal to the tree-elimination dimension of G.*

Proof. The proof uses bottom-up induction on the recursion tree. Let $\bar{C}_1,\ldots,\bar{C}_j,\ldots,\bar{C}_r$ be the children of H_l in the recursion tree (if H_l is a leaf, then this family of $\bar{C}_j$s is empty). The graphs $\bar{C}_j$s are defined at an application of step 4 of the procedure tree-elimination. Let $\partial C_1,\ldots,\partial C_j,\ldots,\partial C_r$ be the corresponding sets of vertices defined at the preceding step 3 of the procedure. We inductively assume that the lemma is true for $\bar{C}_1,\ldots,\bar{C}_j,\ldots,\bar{C}_r$. Observe that if H_m is a descendant of or coincides with H_l, then H_m is a descendant of or coincides with one of the $\bar{C}_j$s or it coincides with H_l. So by the induction hypothesis we only have to prove that the treewidth of H_l is at most equal to the maximum of the treewidths of the graphs $\bar{C}_1,\ldots,\bar{C}_r$ and of the degree of v_{H_l} in H_l. To prove this, consider optimal tree-decompositions $(\{X_i^j: i \in I_j\}, T^j)$, $j = 1,\ldots,r$, for each one of the graphs $\bar{C}_j$, $j = 1,\ldots,r$. By Lemma 3, each of the sets ∂C_j, $j = 1,\ldots,r$ is contained in one node of the corresponding tree-decomposition.

Construct now a tree-decomposition $(\{X_i: i \in I\}, T)$ for H_l in the following way: I is the disjoint union of the $I_1,\ldots,I_j,\ldots,I_r$ plus a new vertex i_0; let X_{i_0} be the set of vertices comprising the vertex v_{H_l} together with its neighbors in H_l; for $i \in I$ and $i \neq i_0$, let X_i be equal to the corresponding X_i^j; finally T is composed by connecting each T^j, $j = 1,\ldots,r$, to i_0 via the vertex k for which X_k^j contains ∂C_j. It is easy to see that this is a tree-decomposition with the required properties. □

Given an edge $\{v,w\} \in E(H_l)$, define its weight with respect to H_l (notationally $\text{weight}_{H_l}(v,w)$) to be the length of the shortest path in G that connects v to w and whose internal vertices precede of v_{H_l} in the ordering π.

Lemma 5. *If G has no chordless cycle of length $> s+2$ (s is a constant positive integer), then no H_l contains a chordless cycle that has length 4 or more and whose edges have weights (with respect to H_l) with total sum $> s+2$.*

Proof. For reasons of linguistic convenience we call the cycles with the properties described in the lemma *bad cycles*. First notice that if $H_{m'}$ is a descendant of or coincides with H_m in the recursion tree, then for every edge $\{v,w\} \in E(H_m) \cap E(H_{m'})$,

$$\text{weight}_{H_{m'}}(v,w) \leqslant \text{weight}_{H_m}(v,w). \tag{1}$$

Suppose now, towards a contradiction, that there are nodes of the recursion tree that contain bad cycles and let H_l be one which is closest to the root $G = H_{l_0}$ of the tree. Since the weight of any edge in G with respect to G is 1, it easily follows from the hypothesis about the cycles of G that H_l does not coincide with the root $G = H_{l_0}$. Let H_{l^-} be the parent of H_l in the recursion tree. Also let C be a bad cycle in H_l. By the fact that H_l is a closest to the root node of the recursion tree which contains bad cycles and by Eq. (1), it easily follows that there is in H_l an edge, say $\{v_1,v_2\}$, that does not belong to any ancestor of H_l in the recursion tree. By the way new edges are added during the procedure tree-elimination, it follows that both v_1 and v_2 are adjacent to $v_{H_{l^-}}$ in H_{l^-}. First we claim that all other edges of C except $\{v_1,v_2\}$ belong also to H_{l^-}. Indeed, if there was an edge $\{v_3,v_4\}$ in C different from $\{v_1,v_2\}$ and not in H_{l^-}, then v_3 and v_4 would be neighbors of $v_{H_{l^-}}$ and at least one of them would be distinct from both v_1 and v_2; but if that were the case, C would contain a chord, a contradiction and therefore the claim is true. Consider now the cycle C', which is obtained from C by deleting $\{v_1,v_2\}$ and adding in its place $\{v_{H_{l^-}},v_1\}$ and $\{r_{H_{l^-}},r_2\}$. Using the above claim, it is easy to verify that C' is chordless (otherwise C would also contain a chord). Finally, by the way weights are defined, it follows that

$$\text{weight}_{H_l}(\{v_1,v_2\}) \leqslant \text{weight}_{H_{l^-}}(\{v_{H_{l^-}},v_1\}) + \text{weight}_{H_{l^-}}(\{v_{H_{l^-}},v_2\}). \tag{2}$$

Thus, C' is a bad cycle in H_{l^-}, a contradiction. □

Lemma 6. *If G has no chordless cycle of length $> s+2$ (s is a constant positive integer) then no H_l contains an edge of weight (in H_l) $> s$.*

Proof. The weight of an edge in G with respect to G is 1, so the lemma is true for the root G. Let, towards a contradiction, H_l be closest to the root node of the recursion tree that contains an edge with weight $> s$ and suppose that this edge is $\{v_1, v_2\}$. H_l does not coincide with the root G of the recursion tree, so let H_{l-} be the parent of H_l. It easily follows from Eq. (1) in the proof of the previous lemma that $\{v_1, v_2\}$ does not belong to H_{l-}. Therefore, from the way new edges are added during the procedure tree-elimination, it follows that both v_1 and v_2 are adjacent to $v_{H_{l-}}$.

Now observe that by the definition of the procedure tree-elimination, v_1 and v_2 are connected by a path in H_{l-} whose internal vertices are not adjacent to $v_{H_{l-}}$. Let P be the shortest such path. Since the edge $\{v_1, v_2\}$ does not belong to H_{l-}, P has length at least 2 and, of course, the sum of weights of its edges is also at least 2. Add now the edges $\{v_{H_{l-}}, v_1\}$ and $\{v_{H_{l-}}, v_2\}$ to P. We thus obtain a chordless cycle in H_{l-} of length at least 4 which, by Eq. (2), has total sum of weights $> s+2$. This contradicts the previous lemma. □

Lemma 7. *If G has no chordless cycle of length $> s+2$ (s is a constant positive integer) then the tree-elimination dimension of G is $\leqslant$ the s-elimination dimension of G.*

Proof. Let π be an elimination ordering of the graph and let $(H_l)_{l\in L}$ (respectively, $(G_i)_{i\in I}$) be the family of graphs generated when we apply tree-elimination (respectively, s-elimination) to G according to π. Consider an arbitrary element H_l of the family $(H_l)_{l\in L}$ and suppose that the vertex v_{H_l} is the ith vertex in π (i.e. $v_i = v_{H_l}$). It suffices to prove that the degree of v_{H_l} in H_l is at most equal to the degree of $v_i = v_{H_l}$ in G_i. For this, it is enough to show that H_l is a subgraph of G_i. Since v_{H_l} is the first vertex in π that is contained in H_l and since G_i contains $v_i = v_{H_l}$ and all vertices after v_i in π, it follows that $V(H_l) \subseteq V(G_i)$. The fact that $E(H_l) \subseteq E(G_i)$ follows immediately from Lemma 6 and the way new edges are added during s-elimination. □

Proof of Theorem 16. By Lemmata 4 and 7, we conclude that

treewidth of $G \leqslant s$-elimination dimension of G.

The other direction immediately follows from Theorem 3 and the obvious fact that the elimination dimension of a graph is at least equal to its s-elimination dimension. □

The following is the conclusion of this section:

Theorem 17. *The treewidth of a graph with no chordless cycles of length $> s+2$ incremented by 1 is equal to its monotone search number for an inert fugitive with speed s.*

Proof. It is immediate from Theorems 15 and 16. □

Open problem. The question of whether "recontamination" may help an inert fugitive with speed a constant $s > 1$ is open.

Acknowledgements

We thank Christos Papadimitriou and Paul Spirakis for their patience to hear us talk about fugitive searching and for their valuable suggestions. We thank Moti Yung, who during a short visit to Patras revived our interest for search games.

References

[1] R. Anderson and E. Mayr, Parallelism and greedy algorithms, *Adv. Comput. Res.* **4** (1987) 17–38; see also: A P-complete problem and approximations to it, Tech. Report, Dept. Computer Science, Stanford University, California, 1984.

[2] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability (a survey), *BIT* **25** (1985) 2–33.

[3] D. Bienstock, Graph searching, path-width, tree-width and related problems (a survey), *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.* **5** (1991) 33–49.

[4] D. Bienstock, N. Robertson, P.D. Seymour and R. Thomas, Quickly excluding a forest, *J. Combin. Theory Ser. B* **52** (1991) 274–283.

[5] D. Bienstock and P.D. Seymour, Monotonicity in graph searching, *J. Algorithms* **12** (1991) 239–245.

[6] H.L. Bodlaender, Classes of graphs with bounded treewidth, Tech. Report RUU-CS-86-22, Dept. of Computer Science, Utrect University, The Netherlands, 1986.

[7] H.L. Bodlaender, A tourist guide through treewidth, *Acta Cybernet.* **11** (1993) 1–21.

[8] H.L. Bodlaender and R.H. Möhring, The pathwidth and treewidth of cographs, in: *Proc. 2nd Scandinavian Workshop on Algorithm Theory (SWAT) 1990*, Lecture Notes in Computer Science 447 (Springer, Berlin, 1990) 301–309.

[9] H.L. Bodlaender and D.M. Thilikos, Treewidth and small separators for graphs with small chordality, Tech. Report UU-CS-1995-02, Dept. of Computer Science, Utrecht University, Utrecht, The Netherlands, 1995.

[10] R. Breisch, An intuitive approach to speleotopology, in: *Southwestern Cavers*, Vol. VI (Southwestern Region of the National Speleological Society, 1967) 72–78.

[11] R. Dechter, Constraint networks, in: S.C. Shapiro, ed., *Encyclopedia of Artificial Intelligence* (Wiley, New York, 1992) 285–293.

[12] R. Dechter, Directional resolution: the Davis–Putnam procedure, revised, in: *Working Notes AAAI Spring Symp. on AI and NP-Hard Problems* (1993) 29–35.

[13] R. Dechter and J. Pearl, Tree clustering for constraint networks, *Artificial Intelligence* **38** (1989) 353–366.

[14] J.A. Ellis, I.H. Sudborough and J.S. Turner, The vertex separation number of a graph, in: *Proc. 1983 Allerton Conf. on Comunication and Computing* (1983) 224–233.

[15] M. Franklin, Z. Galil and M. Yung, Eavesdropping games: a graph-theoretic approach to privacy in distributed systems, in: *Proc. 34th Annual Symp. on Foundations of Computer Science (FOCS) 1993* (IEEE Computer Soc. Press, Silver Spring, MD, 1993) 670–679.

[16] E.C. Freuder, A sufficient condition for backtrack-free search, *J. ACM* **29** (1982) 24–32.

[17] E.C. Freuder, A sufficient condition for backtrack-bounded search, *J. ACM* **32** (1985) 755–761.

[18] F. Gavril, The intersection graphs of subtrees in trees are exactly the chordal graphs, *J. Combin. Theory Ser. B* **16** (1974) 47–56.

[19] N.G. Kinnersley, The vertex separation number of a graph equals its path-width, *Inform. Process. Lett.* **42** (1992) 345–350.

[20] L.M. Kirousis and C.H. Papadimitriou, Interval graphs and searching, *Discrete Math.* **55** (1985) 181–184.

[21] L.M. Kirousis and C.H. Papadimitriou, Searching and pebbling, *J. Theoret. Comput. Sci.* **47** (1986) 205–218.
[22] L.M. Kirousis and D.M. Thilikos, The linkage of a graph, *SIAM J. Comput.*, to appear.
[23] T. Kloks, Treewidth, Ph.D. Thesis, Utrecht University, The Netherlands, 1993.
[24] A.S. LaPaugh, Recontamination does not help to search a graph, *J. ACM* **40** (1993) 224–245.
[25] A. Mackworth, Constraint satisfaction, in: S.C. Shapiro, ed., *Encyclopedia of Artificial Intelligence* (Wiley, New York, 1992) 276–285.
[26] A. Mackworth and E. Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, *Artificial Intelligence* **25** (1985) 65–74.
[27] D.W. Matula, A min–max theorem for graphs with application to graph coloring, *SIAM Rev.* **10** (1968) 481–482.
[28] N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson and C.H. Papadimitriou, The complexity of searching a graph, *J. ACM* **35** (1988) 18–44
[29] R.H. Möhring, Graph problems related to gate matrix layout and PLA folding, in: E. Mayr, H. Noltemeier and M. Syslo, eds., *Computational Graph Theory, Computing Supplementum*, Vol. 7 (1990) 17–51.
[30] T.D. Parsons, Pursuit-evasion in a graph, in: Y. Alavi and D.R. Lick, eds., *Theory and Applications of Graphs* (Springer, Berlin, 1976) 426–441.
[31] T.D. Parsons, The search number of a connected graph, in: *Proc. 9th Southeastern Conf. on Combinatorics, Graph Theory, and Computing*, Utilitas Mathematica, Winnipeg, Canada (1978) 549–554.
[32] N. Robertson and P.D. Seymour, Graph minors III. Planar tree-width, *J. Combin. Theory Ser. B* **36** (1984) 49–64.
[33] P.D. Seymour and R. Thomas, Graph searching and a minimax theorem for tree-width, *J. Combin. Theory Ser. B* **58** (1993) 22–33.
[34] J. van Leeuwen, Graph algorithms, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science*, Vol. A (Elsevier, Amsterdam, 1990) 527–631.